Dynamic, automatic, first-order ontology repair by diagnosis of failed plan execution

Dynamic, automatic, first-order ontology repair by diagnosis of failed plan execution

Fiona McNeill and Alan Bundy
{f.j.mcneill,a.bundy}@ed.ac.uk
School of Informatics, University of Edinburgh, EH8 9LE, Scotland

We describe , an ontology repair system. In contrast to most ontology *matching* systems, is designed to *repair* an ontology that does not accurately model its domain, rather than to find links between two or more unchanging ontologies. It also works on first-order representations rather than just concept taxonomies, classifications or hierarchies. can make belief revisions, but it more often makes *signature* repairs, *i.e.* changes to the arities, types and numbers of predicates. Unlike traditional ontology matching systems, does not require full access to the ontologies of other agents and works entirely automatically and dynamically. However, it does assume a large measure of pre-existing similarity between the ontologies of interacting agents: it is designed to deal with ontologies that are evolving and that come originally from the same source.

works by analysing failed plan executions to diagnose ontological mismatches and to implement repairs. This process iterates until it reaches either successful plan execution or failed plan formation.

is the first example of a new breed of dynamic, automatic ontology-repair mechanisms, which we believe will be essential to realise the vision of autonomous, interacting agents, such as envisaged in the Semantic Web. Full access to another (potentially rival) agent's ontology is unrealistic for both practical and commercial reasons. Static and interactive matching mechanisms are unrealistic in the context of huge and dynamic populations of agents. Full ontological agreement is pragmatically unrealistic, even where a standard ontology has been agreed, due to evolving versions and local customisation.

Addressing these issues is very challenging. We have made a promising start, but much remains to be done. To make initial progress, we have made many simplifying assumptions. We present encouraging experimental results and an analysis of current limitations to be addressed in future work.

The Problem of Dynamic Representation

It is becoming a commonly accepted fact of life in Artificial Intelligence that semantics are fluid. The way in which one person assigns meaning to a term may not reflect another person's way. Even if people are using the same language and representing the same domain, the words they choose to represent a certain meaning, the granularity of concepts they choose and the structure in which these concepts are organised will inevitably diverge. Moreover, theories of knowledge evolve as they are revealed to be inefficient or incorrect or as they are used in different domains to the one in which they were developed. This evolution could be central (an official source of knowledge is updated) or it could be local (individuals or groups using publicly available knowledge can alter it to suit their own requirements).

The research reported in this paper was supported by EPSRC grant GR/S01771, the funded OpenKnowledge project and an studentship to the first author. We are very grateful to our anonymous reviewers for their constructive and helpful feedback.

In AI, it is becoming increasingly apparent that the concept of a global ontology – a definitive account of what exists – is an impossible ideal. In practise it is often the case that agents have individual ontologies that differ, to a greater or lesser extent, from the ontologies of other agents, and in such cases an ontology must be considered to be a single agent's view or perspective on the world. Agents cannot in general interact successfully with other agents unless they are able to understand the terms which those agents are using by matching them to terms from their own ontologies: this is the role of ontology matching. However, we believe that it is not sufficient to match the words and concepts from one ontology to another, but that the signature of the ontologies must also be matched: representation itself is fluid (Bundy & McNeill, 2006).

The problem we address is of a very different nature to the problem of ontology matching. In order to clarify this point and allow the reader to address the paper with a full understanding of our approach, we define our interpretation of traditional ontology matching (Definition 1) and our approach to ontology repair (Definition 2). Comparison of these two definitions reveals the difference in our approach.

Definition 1 (Traditional Ontology Matching) *For two ontologies, O_1 and O_2, a map σ must be found such that for every theorem ϕ_1 in O_1, if it is mapped by σ to a theorem ϕ_2 in O_2, then ϕ_1 is derivable from O_1 if and only if ϕ_2 is derivable from O_2.*

$$\forall O_1 : O, O_2 : O.\exists \sigma : O_1 \times O_2.\forall \phi_1 : O_1, \phi_2 : O_2. \\ \sigma(\phi_1, \phi_2) \rightarrow (O_1 \vdash \phi_1 \leftrightarrow O_2 \vdash \phi_2) \quad (1)$$

Definition 2 (Ontology Repair: our approach) *For an ontology O, for any statement ϕ that is derivable from O but is discovered to be incorrect according to the world, a map π exists such that either $\pi(\phi)$ is correct according to the world and derivable from $\pi(O)$ or, if this cannot be satisfied, $\pi(\phi)$ is not derivable from $\pi(O)$.*

$$\forall O : O.\forall \phi : O.(O \vdash \phi \wedge \not\models \phi) \\ \rightarrow \exists \pi : (O \mapsto O).(\pi(O) \vdash \pi(\phi) \wedge \models \pi(\phi)) \vee \pi(O) \nvdash \phi \quad (2)$$

These definitions highlight the different problems that our approach and the traditional approach are tackling. Both approaches are subsets of the global problem of misalignment between ontology, but are focused on different aspects of that problem. We briefly elaborate on these differences to ensure that the reader has a full understanding of our aims with which to approach the paper.

- The traditional definition of ontology matching is concerned with linking two ontologies, O_1 and O_2. Our approach is focused only one ontology, O: the ontology of the agent or system, A, which is using to help it communicate despite potential mismatches with the world. The meaning of O is then considered not explicitly in terms of another ontology but in terms of feedback from the world or environment in which A is interacting. In a multi-agent system, this interaction with the world will be interactions with other agents, and mismatches between the ontology O and the world will imply mismatches between O and the ontologies of the agents with which A is interacting. However, we emphasise that this is not intrinsic to our approach: the world could be a physical one, for example. Key to this is that in our approach, ontology repair is done from the point of view of one agent: this agent has only limited access to other agent's ontologies and is using to do its best to have successfully interactions in a complex and only partially understood world. This is opposed to the "God's view" approach often used in ontology matching, where it is assumed that both ontologies are fully revealed and that the matching is done externally to any agent using one or other of the ontologies.
- In Definition 1, the ontologies themselves remain unchanged. A map, σ, is found between them and is applied to the objects in them, but σ does not affect the ontologies themselves. In Definition 2, however, π is discovered in reference to a particular object in O that has been discovered to be false in the world and π is then applied to the ontology itself so that a new version of O, $\pi(O)$ is derived. That is, π is a function between one ontology and a repaired one.
- In Definition 1, the ontology map σ depends only on the two ontologies and applies to all sentences, whereas the ontology repair π is dependent on the sentence ϕ which has witnessed the original ontology's inconsistency.
- Whereas traditional ontology matching is generally maximal, that is, a map between the entire ontologies is sought, our approach is minimal: we only wish to change that part of the ontology that has been directly proved to be incorrect. We therefore have an additional minimality constraint, whereby we are only interested in altering the object ϕ that has been discovered to be untrue and any other objects in the ontology O that are dependent on ϕ; all other objects are unchanged by the matching.

The solution we propose to this problem is therefore designed to work at run-time and only when problems become apparent, making ontology alteration necessary. In large, multi-user systems such as the Semantic Web, one might wish to interact with large numbers of different users in a short space of time and may not be able to predict in advance who these may be: it is therefore impossible for us to match their ontologies before run-time. Each interaction may concern only a small part of our, or their, ontology and we may then never interact with this individual again: it is therefore extremely wasteful to match the entire ontologies. Agents may have commercially sensitive material and may be unwilling to reveal their entire ontologies to unknown agents encountered during interactions, or may find it impractical to do so, in which case matches must be gleaned from those parts of their ontologies that they choose to reveal during interactions.

- We are interested not only in matching the meanings of words but also in matching the representation language in which the ontology is written, which is defined in the signature of the ontology. When an ontology evolves, the shift of the meaning of words is only one aspect of the problem. Another key aspect is that signature objects, such as predicate definitions and the type hierarchy, may also change. For example, arguments may be added to predicates, the types of arguments of predicates may change, and so on. In this work we focus on the issues surrounding the shifting of the representational language of the ontology. There has been a great deal of work done in the field of matching words, albeit not always within the confines of the context we are interested in. However, we believe that the problem of matching different representations has not been addressed to any extent in the literature, and yet is of key importance in the problem of evolving ontologies.

This kind of underlying representational change is a common occurrence in every day life. Consider, for example, the everyday experience of buying something from a slot machine. Imagine that the buyer knows that the item they want costs £5, and so comes prepared with a £5 note. However, on closer inspection, it is discovered that the machine does not take notes but only coins: thus the buyer's expectation that he must have £5 is revised to an expectation that he must have £5 *in coins*. Whilst attempting to pay with coins, the buyer may then discover that the machine does not take the new 50p coins - perhaps it is an old machine. Even some coins that the machine claims to accept are unexpectedly rejected - perhaps they are too worn. It may later be discovered that the machine will accept coins which it is not supposed to accept - for example, foreign coins that are similar enough in

shape and size for the machine to confuse them with legitimate coins. In order to obtain a correct understanding of how to buy something from the machine, the buyer must alter the preconditions he has on buying from the machine. However, this does not merely require a change of belief but also of the representation itself. New concepts have to be developed: "coins excluding the new 50p", "coins that are not too worn to be accepted by this particular machine", "foreign coins that will fool this machine", etc.

As another example, consider the experiment conducted by Andreas diSessa on first-year MIT physics students (diSessa, 1983). Students were asked to imagine a ball being dropped from a height onto the floor, and asked to consider how the energy of the ball changes. Before the ball is released, it has potential energy but no kinetic energy. As it is falling, it has kinetic energy and potential energy, and just before it hits the floor it has kinetic energy but no potential energy. However, as it hits the floor it has neither potential nor kinetic energy. The students had difficulty in accounting for this "missing" energy. The answer was that this energy was stored in the deformation of the ball. However, the students had idealised the ball as a particle with mass but no extent and so the correct answer could not be realised within their representation. In order to see the correct answer, they had to not merely change their beliefs about the ball but also change their *representation* of the ball.

In this paper, we present our system, , which uses a diagnostic theory based on abstraction and refinement techniques to diagnose mismatches between ontologies that were similar but have diverged. is intended to be a component that any agent could use that would increase its chances of engaging in successful interactions, because these interactions may be possible even in situations where ontology mismatches would, without diagnosis and repair, lead to failure. focuses not on alignment between concepts, an area where much work has been done and where there are many other systems that can perform this part of the problem, but on structure alignment, a problem that has received considerably less attention. An agent using is not intending to converge to a more correct version of its ontology – though it could be used in this way if all external agents are similar to each other – but rather assesses alignment needs afresh for each agent it encounters and does not assume any consistency between these other agents. is intended to function in situations where agents have ontologies that may have come from the same or a similar source and have diverged over time, rather than ontologies that are completely disparate.

Our hypothesis is that:

> *it is possible, using dynamic ontology repair, to locate and correct ontological mismatches between agents during run-time, to enable successful communication which would otherwise be impossible. This repair must apply not only to the concepts of an ontology but also to its structure.*

In the next section, we clarify our view of ontologies and detail the kinds of ontologies is designed to work with. We then describe the context in which is designed to work and provide a worked example of how functions. The following section details our diagnostic theory and introduces the theory of abstraction and refinement, and then explain how it forms the basis of the diagnostic algorithm in . We then provide an overview of the architecture and subsystems of . Later, we describe the evaluation we have performed on the system, both to confirm that the theoretical functionality is performed successful on ontology matching problems and to provide statistical analysis of how often real ontological mismatches could be successfully diagnosed and refined by and analysis of mismatches for which this failed. We then examine related work and explain how fits in with other ontology matching systems. Finally, we summarise the paper and draw conclusions.

Ontologies

There is some ambiguity and disagreement over the meaning of *ontology*. Here, we define how we use the term in this paper:

Definition 3 (Ontology) *An ontology consists of two parts:*

- *the* signature, *which describes the representation language in which the ontology is written;*
- *the* theory, *which contains formulae written in the representation language and asserted to be true.*

Ontologies are often considered to be simply hierarchies of concepts (or taxonomies), describing only the kinds of things that can exist and the subtype relations between them. This richer notion of ontology, however, describes not just the concepts and their hierarchical relationships, but also how concepts can be related in a non-hierarchical manner through predicates. These provide a rich language for expressing the domain of the ontology.

Our work primarily focuses on the problem of representational mismatch: that is, mismatches in the *signature*. However, alterations in the signature will normally also entail alterations in the theory, where the particular signature object is instantiated, and these are addressed where necessary. There are many different levels of expressiveness which can be contained in an ontological representation. Traditional ontology representations range from full first-order to more restricted representations such as Description Logics to taxonomies. Higher-order ontological representations are not commonly used as reasoning with them is intractable. Our work focuses on first-order ontologies, though the system works with restricted first-order (discussed in the following section).

Context of

> *"To learn, a learner needs to formulate plans, monitor the plan execution to detect violated expectations, and then diagnose and rectify errors which the disconfirming data reveal."*
> Frederick Hayes-Roth (Hayes-Roth, 1983)

We are concerned with the resolution of the problem of ontological mismatch within a planning context, such as would be necessary for an agent to orchestrate Semantic Web services to reach a goal. An agent forms plans to achieve a goal based on its understanding of the domain, and then attempts to execute these plans through communication with other agents. Planning in complex and dynamic environments is very difficult because any incomplete, incorrect or out-of-date information can cause an inexecutable plan to be developed because the environment is changing while planning is being performed. However, by adopting our approach to ontological repair, these cases of plan execution failure can be considered to be opportunities to learn more about the domain through repairing a mismatched ontology. Executing plans, in our environment, is done by interacting with agents who can perform the necessary tasks of the plan: for example, buying a ticket is performed by successfully interacting with a ticket-selling agent. Information about the cause of failure is extracted from observation of the communication surrounding plan failure, augmented by further communication with the other agents involved. Once the point of failure has been located, repair techniques are implemented to fix the problem, and a new plan is developed using the updated ontology. This plan is more likely to succeed than the previous plan. This procedure is repeated until the goal is successfully reached, or until it becomes impossible to form a plan to achieve the goal from the updated ontology. In , Definition 2 is used retrospectively rather than proactively. That is, does not ensure it is enforced every time an interaction takes place, but if an interaction leads to plan failure then the equation is used to determine the cause of the failure.

We consider that there are three essential elements to creating such a dynamic ontology repair system:

1. the ability to link the relevant information about the underlying ontology to the plan;

2. the ability to use this information to diagnose the exact source of the problem;

3. the ability to select and apply appropriate techniques for altering the ontology.

Of these, the ability to diagnose mismatches forms the heart of the system. The ability to link plan execution details to the ontology which formed the plan and the ability to refine the ontology after diagnosis are also crucial but are less theoretically difficult. We therefore mention these latter abilities in passing later in the paper, but dedicate the bulk of this paper to discussing diagnosis.

In order to determine how to match the excerpts of unknown ontologies revealed via communication to a known ontology, we need to determine what the representation of those ontologies are. Since we are interested in a planning context, we are interested in representations that are expressively rich enough to allow planning. We have therefore developed our diagnostic theory to deal with ontologies written in first-order logic. This expressive power allows us to deal with an interesting version of the problem, partly because it allows scope for many kinds of mismatch and partly because it allows interesting planning. However, in order to limit the difficulty of the problem in the first instance, we decided to exclude existential quantification from our algorithms, with all variables being implicitly universally quantified, and our algorithms therefore cannot be said to work with full first-order logic.

When implementing the diagnostic theory in the system, we needed to find a representation that was first-order but that was also a standard ontology format, and therefore choose to work with ontologies written in (Knowledge Interchange Format)(Genesereth & Fikes, 1992). is a popular full first-order ontological representation format. Choosing as the representation to explore this issue allowed us to examine the problem in a rich environment. ontologies include types, predicates, individuals and axioms. Axioms can be thought of as implication rules that have a conjunction of relations determining when the rule is applicable, and a conjunction of relations describing the situation after the rule has been applied and can therefore describe plan action rules. Much of our research focuses on identifying what potential mismatches could arise between these ontological objects. Since our diagnostic algorithms were designed to deal with first-order logic excluding quantification, we must deal only with ontologies that do not have quantification: this is normally allowed in . Additionally, allows complex class definitions which are not covered by our first-order diagnostic techniques, and we therefore use a restricted version of that uses simple class definitions.

When an ontological mismatch is detected, there is the potential for some complex negotiations between the agents as to which of them ought to refine their ontologies, or, indeed, whether both should. Factors that might be relevant here are whether one of the agents is recognised, perhaps by the community, perhaps by the other agent, to be an authority on the matter; whether one is controlling the situation by, for example, being able to provide something that is required by the other agent; whether either of the agents consider this part of their ontology to be particularly important and are unwilling to compromise it; and so on.

We simplify all these issues by assuming that the planning agent ( ) is willing to take on trust any information that is given to it by another agent. We believe that this assumption is plausible, because, in this scenario, the is interacting with other agents because it wishes them to provide services for it; hence the other agents are in control of the situation. This assumption also makes the situation tractable and allows us to avoid getting side-tracked onto important but tangential issues.

Example 1 (Online travel planning) *The following worked example is intended to illustrate the system as it creates and executes plans, detects and diagnoses failure, repairs its ontologies, then replans recursively until it either constructs a plan that executes successfully or it fails. The domain concerns the submission of the camera-ready copy of an accepted paper to a conference, then making plans to attend the conference. It uses a peer-to-peer, multi-agent system. Various agents assist the paper writer to submit the paper and attend the conference. They interact together to achieve the various goals this entails.*

In this example, the planning agent acts as the paper writer's personal assistant. The plan includes steps in which the other agents perform various services, e.g., accept the submitted paper. If the planning agent's ontology were an accurate model of the world then it would be bound to succeed. So a failure during plan execution signals an ontology failure. For instance, the submission agent might refuse to accept the paper. The traveller must then diagnose the fault in its ontology and repair it. A new plan for the goal is then derived and executed. This new plan may also fail, triggering a further round of diagnosis and repair. This process recurses until either a plan is derived that successfully executes or the planning agent is unable to diagnose or to repair a fault.

The agents involved are as follows:

Planning Agent: *which is responsible for forming and executing the overall plan.*

Publication Agent: *through which the camera-ready copy can be sent for publication in the proceedings.*

Registration Agent: *which can register conference attendees.*

Accommodation Agent: *which can book accommodation for a conference delegate.*

Paper Conversion Agent: *which is able to convert papers into different formats.*

The planning agent forms the following plan from its initial ontology [1]*:*
(SendPaper Researcher My-Paper.ps Ai-Conf),
(Register Researcher Ai-Conf Registration-Cost),
(BookAccom Researcher Ai-Conf Accommodation-Cost).

The first action, SendPaper, *fails to be executed. There was some questioning before the failure occurred, which included a query from the publication agent:* (Format My-Paper.ps Pdf-Paper). *The had not realised that this was a precondition to the action but the fact that a question was asked about it before the action could be performed means that it is obviously important. The must determine whether this statement is true, and since My-Paper.ps is in Ps format and not Pdf, it replies to the publication agent that this is not true. The publication agent then fails to perform the action. The can then determine that this ought to be a precondition to the action and that it ought to change its rule concerning the action to include it. It now plans again. This additional precondition of the SendPaper action is not initially true, so a new action must be included to make it true. The new plan is:*
(Convert-Paper Researcher My-Paper.ps My-Paper.pdf)
(SendPaper Researcher My-Paper.pdf Ai-Conf),
(Register Researcher Ai-Conf Registration-Cost),
(BookAccom Researcher Ai-Conf Accommodation-Cost).

The first and second action are now executed successfully. The third action, Register, *fails, following a surprising question:* (Money PA Credit-Card ?Amount), *where* PA *is the . The predicate* Money *matches a precondition that* PA *was expected to be asked about:* (Money PA 100)*; however, the surprising question has an extra argument concerning type of payment. The discovers what the class of this extra argument is (if it does not currently have this information, it can question the registration agent) and alters its definition of the* Money *predicate to include this extra argument. Every instance of this predicate must also be changed. Before the can make plans using this, it must verify that* Credit-card *payment is indeed possible.*

After this change has been made, the replans to form the following plan:
(Register Researcher Ai-Conf Credit-Card Registration-Cost),
(BookAccom Researcher Ai-Conf Credit-Card Accommodation-Cost).
These actions are performed successfully.

This worked example is part of a larger example which has been implemented in the system and successfully executed.

Determining Potential Mismatches

The diagnostic process depends on a systematic analysis of possible mismatches in the given representation - in our case, first-order logic. There are two steps in producing a successful diagnostic algorithm: firstly, determining the space of possible mismatches for the representation and secondly, developing techniques that can use the available information to search through this space and find a precise diagnosis. The latter process depends on the context in which interactions are taking place and on what this available information about the mismatch happens to be in a given situation, and in many cases it may be that a precise diagnosis cannot be found. The former process, however, must be a precise and methodical analysis of the problem. If there are situations in which certain mismatches cannot be precisely diagnosed (as is the case in the context of ; these situations are discussed later in the paper) this is because developing diagnostic techniques that can opportunistically take advantage of the available knowledge is difficult and there may be cases where the available information is simply not enough to narrow the mismatch down to a specific possibility. Such situations, however, do not point to a flaw in the space of possible mismatches.

In this section, we define what we mean by first-order ontologies and then present an analysis of possible mismatches between such ontologies. This analysis is generic and can be applied to any situation in which mismatches between such first-order ontologies are diagnosed. We then describe the diagnostic techniques we build on top of this to perform diagnosis in the specific context of our system.

[1] The representation used in this example is : the first element of a tuple represents the predicate and the following elements, the arguments. Arguments beginning with a question mark are variables; otherwise, they are constants.

Definition 4 (First-Order Ontology) *We consider first-order ontologies that contain:*

- *A* **signature**, *containing:*
- *Definitions of predicates: their names, arity and number of arguments. Unary predicates are class definitions.*
- *Details of relationships between predicates. In the case of the unary predicates, these relationships define a class hierarchy.*
- *A* **theory**, *containing:*
- *Instantiations of the predicates defined in the theory.*
- *Action rules, which define the way in which performing actions as part of a plan can alter the truth values of facts in the theory. Action rules have preconditions, stating which facts must be true or false in the theory for them to be applicable, and effects, stating what facts are altered as a result of the action.*

This is described in Figure 1.

<predicate>::=<pred_name> <args> <super_pred>
<args>::=<variable_id><type>
| <variable_id><type> & <args>
| []
<type>::=<type_name>, <super_type>
<super_type>::=<top>
| < type >
<super_pred>::=<top>
| < pred_name >
|[]
<rule>::=<conditions>→<conditions>
<conditions>::=<predicate>
| <predicate> & <conditions>
| []
<pred_name>::= term
<type_name>::= term

Figure 1. BNF description of the signature of ontologies on which the diagnostic algorithms are based. uses ontologies written in a form of that has been slightly simplified so as to conform with this description.

We believe this is a standard definition of a first-order ontology with the exception of the inclusion of action rules, which describe the necessary conditions for actions to be performed and the effects of that action. Since we are considering diagnosis in a planning environment, we need to include these in our definition of an ontology. The mismatches between ontologies without action rules would still be fully described by our analysis of possible mismatches but our diagnostic techniques could not be applied in the way we have envisaged.

Note that the BNF in Figure 1 describes the signature of appropriate ontologies and not the theory. The signature details the way in which ontological objects such as predicates must be defined; the theory contains specific instantiations of these signature objects. Because theory objects are specific instantiations of these signature objects, they do not need to contain all the general information about these objects. For example, Figure 1 tells us that a predicate definition consists of a predicate name, a number of args which must all has a variable identifier and a type for that variable, and a super-predicate (which could be null) which details how the predicate fits into the predicate hierarchy. In practice, this may look like the following:

$$(Define\text{-}Predicate\ \ Paper\ \ (?Title\ \ ?Author)\ \ (Title\ \ ?Title)\ \ (Author\ \ ?Author)\ \ Written\text{-}Item) \quad (3)$$

The first item in the definition is the predicate name, the second item is a list of the variable identifiers, whose types are detailed by the third and fourth arguments, and the final argument is the super-predicate. However, when this predicate is instantiated to create a theory object, general information such as the types of the arguments and the super-predicate do not need to be detailed: these can be deduced by reference to the signature definition of the object. Instead, the specific instantiations of the variables must be given, and the types of these instantiations must agree with the type restrictions given in the signature. So, for example, an instantiation of this predicate may be:

$$(Paper\ MyPaper\ Smith). \quad (4)$$

The examples given below refer to mismatches in signature objects; however, for the sake of conciseness, we do not always give the full definitions of them as shown in the example predicate definition above. Instead, for these particular examples, we adopt a shorthand version of the definition which gives the pertinent information and appears similar to a theory instantiation of the predicates. For example, the definition of *Paper* in Equation 3 may be given: (*Paper* ?*Title* ?*Author*), which is intended to signify that the predicate *Paper* has two arguments of type *Title* and *Author*. Such variable names would normally just be place holders and the type definitions would have to be referred to. However, it is very common to use the names of the types for these placeholder names and, for the sake of convenience, in the examples below we assume that this is the case and therefore omit the actual type information. We do not give information about the super-predicate unless it is pertinent to the mismatch we are describing.

Abstraction and Refinement

The reasons that ontologies are altered are very often concerned with the fact that too much or, more often, too little detail has been included in the ontology. This creates the need to remove detail - performing an *abstraction* of the ontology, or add detail - performing a *refinement*. Notions of abstractions are often quite vague, but the theory of abstracting first-order ontologies was formalised by Walsh and Giunchilgia (Giunchiglia & Walsh, 1990), who created a theory of abstraction through observing and categorising large numbers of examples of abstraction. Since abstractions should change the theory and not the logic (except in the case of reducing a unary predicate to a nullary predicate, which could be seen as reducing first-order logic to propositional logic),

the problem of building abstractions can be reduced to the problem of deciding on a suitable matching of atomic formulae. They developed four categories into which abstractions of objects in first-order ontologies can fall:

1. **Predicate abstractions**
matching predicate names in some uniform way:e.g.,
(Bottle ?X), (Cup ?X) map onto *(Container ?X)*.

2. **Domain abstractions**
matching constants and function symbols in some uniform way: e.g.,
(Prime 3), (Prime 5) map onto *(Prime Oddnumber)*.

3. **Propositional abstractions**
dropping some or all of the arguments to predicates: e.g.,
(Abelian GroupA), (Abelian GroupB) map onto *(Abelian)*.

4. **Precondition abstractions**
matching some of the atomic formulae onto true or false: e.g.,
(Has Ticket Me) $\rightarrow$ *(Can-Travel Me)* maps onto *(Can-Travel Me)*

We have developed corresponding refinement categories through inverting these abstraction categories:

1. **Predicate refinement**
A single predicate is split into one or more subtype predicates, e.g.,:
(Money ?Amount) maps onto *(Dollars ?Amount), (Euros ?Amount), (Sterling ?Amount)*, etc.

2. **Domain refinement**
The type of an argument is divided into one or more subtypes, e.g.,:
(Money ?Amount European) maps onto *(Money ?Amount Euros), (Money ?Amount Sterling), (Money ?Amount Krona), etc.*

3. **Propositional refinement**
The arity of a predicate is increased, e.g.,:
(Money ?Amount) maps onto *(Money ?Amount Dollars), (Money ?Amount Sterling), etc.*.

4. **Precondition refinement**
A precondition is added to a rule, e.g.,:
(Has Money ?Agent) $\rightarrow$ *(Has Item ?Agent)* maps onto
(Has Money ?Agent) $\wedge$ *(InStock Item Shop)* $\rightarrow$ *(Has Item ?Agent)*

Naturally, not all possible mismatches in first-order techniques are concerned with abstraction or refinements, but these categories describe a large number of mismatches that are encountered and provide a basis for developing a full theory of the space of possible mismatches between first-order ontological objects.

Types of Mismatches

A mismatch between ontological objects must exist between ontological objects of the same type (otherwise the two objects are not mismatched, they are simply different). Since each ontological object has only a certain number of attributes, and the way in which objects are mismatched must be exhibited as a mismatch between one of these attributes, there are only a certain number of ways in which mismatches between each kind of ontological object can occur. Since there are only a few kinds of ontological objects, we can detail precisely the possible ways in which two ontological objects could be mismatched. Note that we do not consider compound mismatches caused by combinations of different mismatches.

1. **Predicates:** Predicates have names, arities and types of arguments.

- **Changing a predicate name:** This is the case in which a predicate is changed to one with a different name but with the same arguments:

$$(p\ \vec{x}) \mapsto (q\ \vec{x}), p \neq q \tag{5}$$

This can be broken down into four situations:
predicate refinement: $p \subset q$: for example,
$(Paper\ ?Paper) \mapsto (PdfPaper\ ?Paper)$,
predicate abstraction: $p \supset q$: for example,
$(PdfPaper\ ?Paper) \mapsto (Paper\ ?Paper)$,
$p \sim q$: for example, $(Paper\ ?Paper) \mapsto (Article\ ?Paper)$,
$\mathbf{p \nsim q}$: for example,
$(Paper\ ?Paper) \mapsto (Conference\ ?Paper)$. [2]

- **Changing the arity:**

$$(p\ \vec{x}) \mapsto (p\ \vec{x}\,y)\ \ or\ \ (p\ \vec{x}\,y) \mapsto (p\ \vec{x}) \tag{6}$$

Here, there are two situations.
For example, **propositional refinement:**
$(Paper\ ?Title\ ?Author) \mapsto (Paper\ ?Title\ ?Author\ ?Format)$
or **propositional abstraction:**
$(Paper\ ?Title\ ?Author\ ?Format) \mapsto (Paper\ ?Title\ ?Author)$

- **Changing the types of arguments:**

$$(p\ \vec{x}\,y) \mapsto (p\ \vec{x}\,z), type(x) \neq type(y) \tag{7}$$

As with the change of name of a situation, this breaks down into four cases:
domain refinement: $type(x) \subset type(y)$: for example,
$(Submit\ ?Paper) \mapsto (Submit\ ?Item)$,
domain abstraction: $type(x) \supset type(y)$: for example,
$(Submit\ ?Item) \mapsto (Submit\ ?Paper)$,
$type(x) \sim type(y)$: for example,
$(Submit\ ?Paper) \mapsto (Submit\ ?Article)$
$type(x) \nsim type(y)$: for example,
$(Submit\ ?Paper) \mapsto (Submit\ ?Quotation)$.
The first two cases indicate that the argument has been made more or less general, the third does not indicate a semantic change but merely a change in label and the fourth indicates a complete alteration of the argument.

- **Switching the arguments:**

$$(p\ \vec{x}\,y\,z) \mapsto (p\ \vec{x}\,z\,y) \tag{8}$$

For example:
$(Paper\ ?Title\ ?Author\ ?Format) \mapsto$
$(Paper\ ?Title\ ?Format\ ?Author)$

- **Predicate relationships:** Relationships between predicates constitute a hierarchy. Each predicate can be associated

[2] where $\subset$ represents an abstraction from p to q, $\supset$ represent a refinement from p to q, $\sim$ indicates that p and q are semantically equivalent even though they are labelled differently and $\nsim$ indicates that there is no known relationship between them.

with an optional super-predicate, which indicates its immediately more general predicate. Predicates in general do not need to have such super-predicates; the exception is types, which must have a super-type. The only way in which this can be changed is for the super-predicate of a predicate to be altered, added or removed.

$$super\_pred(p) = q \mapsto super\_pred(p) = r \quad (9)$$

There are six ways in which this can happen:
$q \subset r$: for example, the super-predicate *written-item* in Equation 3 becomes *item*,
$q \supset r$: for example, a super-predicate *item* becomes *written-item*,
$q \sim r$: for example, a super-predicate *written-item* becomes *document*,
$q \nsim r$: for example, a super-predicate *written-item* becomes *place*,
$q = \emptyset$: that is, a predicate that had no specified super-predicate is given one, r,
$r = \emptyset$: that is, a predicate with a super-predicate q is matched to one with no super-predicate.
If p is unary (i.e., a type) then neither $q = \emptyset$ or $r = \emptyset$ are possible.

2. **Action Rules:** The only aspect of action rules that can change is through the addition or removal of preconditions of effects. Any alteration to the preconditions or effects themselves would be covered by the predicate changes described above.

- **Changing preconditions:**

A precondition is removed - **precondition abstraction**:

$$(\Phi \wedge (p\ \vec{x}) \rightarrow \Theta) \mapsto (\Phi \rightarrow \Theta) \quad (10)$$

where Φ is a set of preconditions, Θ is a set of effects and $(p\ \vec{x})$ represents an extra precondition, for example:

$$\begin{array}{l}(Submitted\text{-}Paper\ ?Paper\ ?Author\ ?Conference) \\ \wedge(Format\ ?Paper\ ?Pdf) \\ \rightarrow (Registered\ ?Author\ ?Conference)\end{array} \quad (11)$$

$\mapsto$

$$\begin{array}{c}(Submitted\text{-}Paper\ ?Paper\ ?Author\ ?Conference) \rightarrow \\ (Registered\ ?Author\ ?Conference)\end{array}$$

or a precondition is added - **precondition refinement**:

$$(\Phi \rightarrow \Theta) \mapsto (\Phi \wedge (p\ \vec{x}) \rightarrow \Theta) \quad (12)$$

: as above but in reverse.

- **Changing effects:** An effect is removed:

$$(\Phi \rightarrow \Theta \wedge (p\ \vec{x})) \mapsto (\Phi \rightarrow \Theta) \quad (13)$$

For example:

$$\begin{array}{c}(Submitted\text{-}Paper\ ?Paper\ ?Author\ ?Conference) \rightarrow \\ (Registered\ ?Author\ ?Conference)\end{array}$$

$\mapsto$

$$\begin{array}{c}(Submitted\text{-}Paper\ ?Paper\ ?Author\ ?Conference) \rightarrow \\ (Registered\ ?Author\ ?Conference) \wedge \\ (Accommodation\text{-}Place\ ?Author\ ?Place)\end{array}$$

or an effect is added:

$$(\Phi \rightarrow \Theta) \mapsto (\Phi \rightarrow \Theta \wedge (p\ \vec{x})) \quad (14)$$

3. **Individuals:** Individuals may change names:

$$a \mapsto b \quad (15)$$

For example, $Smith \mapsto Brown$.
Additionally, individuals may change their type:

$$type(a) = \tau_1 \mapsto type(b) = \tau_2 \quad (16)$$

Again, there are four possibilities:
$\tau_1 \subset \tau_2$: for example,τ_1 = author, τ_2 = person,
$\tau_1 \supset \tau_2$: for example,τ_1 = person, τ_2 = author,
$\tau_1 \sim \tau_2$: for example,τ_1 = author, τ_2 = writer,
$\tau_1 \nsim \tau_2$: for example,τ_1 = author, τ_2 = hotel.

In addition to changing ontological objects that are already present in the ontology, whole ontological objects can be added or removed.

We do not, at this stage, make any comment on how likely these changes are to occur nor on how it would be possible to automatically diagnose such changes: such things are discussed in the section on diagnosis. Our intention here is to develop a theory of the way in which ontologies written according to Definition 4 can be changed. We claim that every possible change to such a theory must fall into one of the categories described above. We therefore use this theory as a basis for our diagnostic algorithm.

Diagnosis and Repair

We consider the major contribution of our work to be the ability of to diagnose and repair ontological mismatches discovered during agent communication. The analysis of potential mismatches discussed above provides a systematic method for categorising mismatches in order to determine what the appropriate repair is. In some domains, this theory could be used to fully diagnose all potential mismatches. However, in the domain in which operates - in an open, distributed multi-agent system, where agents may be willing to cooperate only to a certain level and do not wish to be fully open - we will often face the problem of incomplete information. Additionally, the way in which detects problems - through the unexpected failure of plan execution - means that certain kinds of ontological mismatch will never be highlighted. The ability of to link known ontological objects to new objects that are revealed during communication can, in some cases, depend on the linguistic ability to link two words. takes a simple approach to this problem, depending on links between these words being stated explicitly in the ontology, largely through the predicate and type hierarchy. The problem of semantic mismatch between words or hierarchies has been widely studied and efficient solutions are available. A version of that could truly cope in the real-world scenarios we have envisaged would require links to such systems so that it could perform the semantic word matching in a more realistic manner. However, the work we present here focuses on the structural aspects of the problem.

The Diagnostic Algorithm

Having outlined the ways in which ontological mismatch may occur, we now describe how we can apply this theory to develop a diagnostic algorithm that operates in the context of the system.

Diagnostic Assumptions. In order to make the problem tractable in the first instance, and to focus the research on the problems of diagnosis and repair rather than on other unrelated problems, we have made various simplifying assumptions, which we explain here.

- The system is designed to deal with errors on a case-by-case basis. We do not need to make the assumption that failure is caused by exactly one error; we can identify one error, fix it, replan, encounter failure again and then diagnose a second error. This is not the most efficient way to deal with such a situation but is reasonable if we believe that in most cases, a particular failure will be caused by a particular error. However, the nature of this approach to multiple errors means that we are forced to assume that these errors are independent, and that dealing with them one by one will always lead towards a more correct ontology. This is not always going to be a valid assumption in a real world situation, and the system will fail if it encounters compound mismatches that cannot be diagnosed as a series of individual mismatches.
- We assume that the only information we can get from other agents is that which is revealed by the questions they put to the , and by their answers to direct questions put by the to them.
- We make various simplifying assumptions about the agents with whom interaction takes place. We assume that they are helpful and honest, that they will always perform actions for one another if they are able to, that they are only capable of communicating by exchanging messages and that they share a common protocol: that is, the format of the messages they send is the same, it is only in the content of the messages that mismatches occur. The problem of mismatched protocols is also important but we do not deal with this in our work.

Linking Plan Failure to Ontological Mismatches. The means of detecting that an ontological mismatch has occurred, and of determining what that ontological mismatch may be, is through agent communication. The only information available about mismatches is that which can be gleaned from observation of past agent communication, from forming appropriate questions and putting them to the appropriate agent, and from analysis of the ontology.

The kinds of mismatches that we are primarily interested in are signature mismatches, where the definitions of signature objects differ between agents; these potential mismatches have been outlined earlier in the paper. However, agents do not normally communicate using abstract signature definitions of objects. Agent communication tends to consist instead of fully or partially instantiated theory objects such as described in Equation 4; the would not have direct access to other agent's signature definitions, such as that described in Equation 3. However, such theory objects, since they are based on signature definitions, can reveal information about those signature definitions that indicate to the that there is a signature mismatch. For example, if the has a signature definition of *Paper* according to Equation 3 and, during communication with another agent, is passed a message: *(Paper Banana Smith)*, it can infer that there is a signature mismatch because it knows that *Banana* is of type *Fruit*, which is distinct from type *Author* that the expected such arguments to have. Equally, if the receives a message or query *(Paper MyPaper Smith Ai-Conf)*, it can again infer a signature mismatch as the number of arguments of the predicate do not tie in with its definition of the predicate.

The role of the diagnostic algorithm is therefore to allow the to link information gleaned from theory objects that lead to communication failures to one of the signature mismatches defined in the space of possible mismatches defined earlier in the paper. We emphasise again that this diagnosis is applied pragmatically: we only attempt to infer such information and form diagnoses and repairs on the basis of that inference if communication has failed. In normal circumstances, we make a pragmatic assumption that other agent's ontologies match ours: though we know this to be unlikely for the whole ontology, this assumption allows the to interact with the world and form plans for achieving goals. In situations where this assumption leads to failure, we are then forced to determine what particular mismatch caused this failure, and can then proceed to interact in a useful way with the world.

is intended to be accessed by a , which is attempting to achieve a goal. First, the forms a plan to reach the goal based on its understanding of the domain in which its operating, expressed in its ontology, where each plan step involves invoking services from other agents in the network. It then attempts to execute the plan through communication with these agents. If its understanding of the domain and of the circumstances under which these services can be performed matches those of the 's ( ), the goal should be reached successfully (since we are ignoring complicating factors such as unhelpful agents and network problems). However, if there are ontology mismatches between the and any of the s with which it must communicate, and these mismatches are pertinent to the particular service to be performed, this may result in unexpected behaviour on the part of the and, potentially, failure to provide the service and hence plan execution failure. In this situation, the must invoke to determine what the source of the problem is.

The questions put to the by the s provide the richest source of information. In many cases, the source of the ontological mismatch can be directly identified from these questions, through the information these questions reveal about the ontology of the .

We have developed the notion of *surprising questions*, which are questions put by a to the which do not directly pertain to the preconditions of the action, as far as the understands them. So, for example, if the were asking an to perform an action that it believed was described by the rule in Equation 11, it would expect to be asked about paper submission and format, since these are identified as

preconditions of the rule. If it were asked a question:
(*Money Author* ?*Amount*)

this would be *surprising*, because nothing in its description of the rule leads the to suppose this is pertinent. Also, a question such as:
(*Submitted-Paper Your-Paper Ai-Conf Smith*)

would be surprising: although *Submitted-Paper* is an expected precondition, it is expected that the second argument is of type *Author* and the third of type *Conference*, whereas the identifies the second argument in this question, *Ai-Conf*, as of type *Conference* and the third argument, *Smith*, as of type *Author*.

These surprising questions can provide information about where ontologies between two agents may differ. When a question is put to it by the , it answers it as best as it can and then compares this question against its list of preconditions to see if it exactly matches one of these. If it does not, it is flagged as a surprising question. No further action is taken at this stage, but if plan failure occurs then the surprising questions are referred to. Of particular interest are surprising questions that are asked immediately before plan failure occurs, referred to as Relevant Surprising Questions (s). Note that not all preconditions will be checked with the : some, the can determine without checking. Such preconditions can present problems in diagnosis as the lack of communication concerning them means that it is very difficult to deduce anything about them.

Very often, the information contained in a surprising question is enough on its own to find the source of the problem. For example, if the question contains an instance of a recognised predicate, but with an unexpected arity, or with an argument with an unexpected type, then it is clear that the expectations of this predicate are to blame. However, sometimes it is discovered, possibly through information revealed in a surprising question, that there is a problem with a fact, believed by the to be true, but believed by the to be false. If there is no signature mismatch with respect to this fact – *i.e.*, the predicate expressing this fact is correct according to the 's signature but simply instantiated in a way that the did not expect – then linking the plan execution failure to a flaw in the underlying ontology is more difficult. It must be established how the fact came to be believed. The fact may be present in the original ontology, or it may have been added to the ontology because it was believed to have been the effect of a previously performed action. In the former case, our policy is to remove the fact from the ontology, preferring the 's belief that it is incorrect to the 's belief that it is correct, since the is the one that is particularly equipped to perform the service and thus, we assume, better informed concerning information that relates to it. In the latter case, we must examine the faulty fact to discover how it came to be believed. We refer to the algorithm that diagnoses these incorrect facts as the Shapiro algorithm, because it is loosely inspired by Shapiro's work on algorithmic program debugging (Shapiro, 1982). We have not attempted to follow Shapiro's algorithms closely but have merely used his ideas as an inspiration. This algorithm is discussed later in this section.

The examples we give below are examples of the objects that will be revealed during communication, and therefore these objects are always theory objects. If a theory object indicates the presence of a signature mismatch, the must use inferred information and information derived from questioning the to determine how to alter its own signature appropriately.

The algorithms used in the diagnostic processes are illustrated below as flow charts. Figure 2 illustrates the top-level decision procedure, and Figures 3 and 4 illustrate sub-algorithms that are called once the higher-level decisions have been made.

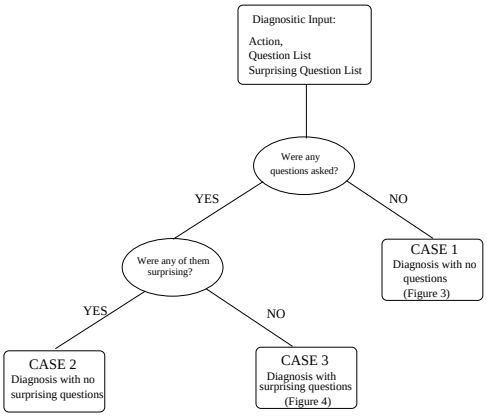


Figure 2. Top Level Decision Making

Figure 2 separates the diagnosis into three separate cases: those where no questions were asked between the request to perform the action and the failure; those where questions were asked but none of them were surprising; those where there were questions asked and at least one of them was surprising.

1.Failure immediately after a request to perform an action has been made.

Example 2 *The forms a plan:*
(Send-Paper Researcher My-Paper.ps Ai-Conf),
(Register Researcher Ai-Conf Registration-Cost),
(BookAccom Researcher Ai-Conf Accommodation-Cost).

The then consults its ontology as to the appropriate agent to approach - in this case the publication agent *(-) - and contacts it. The conversation is as follows:*

: (Send-Paper Researcher My-Paper.ps Ai-Conf),
- : no

This situation is illustrated in Figure 3.

In this situation, it can be difficult or even impossible to diagnose what the cause of failure is because the amount of information we have access to is quite limited. The fact that failure has occurred without any questioning is certainly helpful information, but it reveals less about the ontology of the other agent than the other two situations.

We identify three situations in which this may occur.

- **i) The has been asked to perform a task it is not able to.**

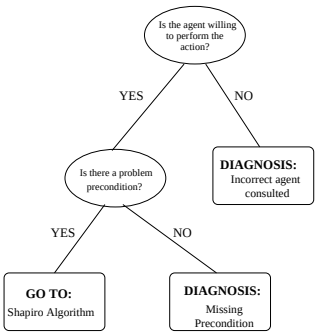


Figure 3. Diagnosis when no Questions have been Asked

We can eliminate this option by querying the agent as to whether it is capable of performing the action. If not, we must remove the information that this agent can perform the action from our ontology and replan.

Example 3 *In fact, a* submission agent *should be contacted rather than a publication agent.*

- **ii) One of the preconditions for the action, that we believed to be true, is not true.**

The and the both have the following preconditions for the action:
$(p\ \vec{x}) \wedge \phi$,
where $(p\ \vec{x})$ indicates a predicate (p) with some number of arguments $\vec{x}$ and ϕ indicates some number (≥ 0) of additional preconditions.
However, we also have:

$$O_{PA} \models (p\ \vec{x})$$
$$O_{SPA} \not\models (p\ \vec{x})$$

That is, $(p\ \vec{x})$ is true according to the ' ontology but not according to the ' ontology.
We can investigate this option by querying the other agent as to its beliefs of the truth values of our preconditions. If we discover a precondition that it does not believe to be true, then this is a likely cause of the failure. We then need to consider why we came, incorrectly, to believe this precondition to be true by using the Shapiro algorithm.

Example 4 *The has the following action rule describing how papers are submitted:*
(And (Is-Open ?Conf-Submission) (Has-Paper ?Agent ?Paper)) ⇒ (And (Accepted-Paper ?Agent ?Paper ?Conference)).
The also has the fact (Is-Open Ai-Conf-Submission) in its ontology (where Ai-Conf is the relevant conference).
The conversation continues from Example 2 as follows:
: (Is-Open Ai-Conf-Submission)
- : no
Therefore, the and the - disagree about this fact.

- **iii) We are missing a precondition, and that precondition is not currently fulfilled.**

The 's preconditions for the action are: ϕ,
whereas the 's preconditions are: $(p\ \vec{x}) \wedge \phi$.
If it happens, coincidentally, to be the case that
$O_{SPA} \models (p\ \vec{x})$
then the communication will proceed as expected and this mismatch will not be identified. However, since failure has occurred, it can be determined that $O_{SPA} \not\models (p\ \vec{x})$ is actually the case. The problem in this situation is discovering what $(p\ \vec{x})$ is. Information about this unexpected precondition cannot be gleaned from previous communication, because at this stage no questions have been asked: this is clearly a precondition that the does not need to verify with the . This case can be distinguished from case 2 by verify the perceived truth value of every precondition in ϕ. If we have:
$\forall (q\ \vec{x}) \in \phi . O_{SPA} \models (q\ \vec{x})$,
we can deduce the presence of an additional precondition $(p\ \vec{x})$ in the 's preconditions, but we cannot infer anything about what the predicate is. Nor can we query the agent about what this precondition is, since we have no basis to guess what it might be, and we cannot assume that the other agent will reveal its complete set of preconditions for this action. Thus our only recourse during repair is to flag this rule as incomplete and not use it during planning.

Example 5 *The has the following action rule describing how papers are submitted:*
(Has-Paper ?Agent ?Paper) ⇒ (Accepted-Paper ?Agent ?Paper ?Conference)
whereas the - has a different version of the rule (though the cannot know this):
(And (Is-Open ?Conf-Submission) ∧ (Has-Paper ?Agent ?Paper)) ⇒ (Accepted-Paper ?Agent ?Paper ?Conference)
The can ascertain that its single precondition is correct according to the -- and can therefore deduce that the - must have an additional precondition in its rule. However, the cannot discover what this precondition is.

2.Failure after a surprising question.

This situation is illustrated in Figure 4.

We can categorise surprising questions in the following ways:

(a) The name of the predicate in the surprising question matches the name of a precondition:

(i) The number of arguments of these two 'matching' predicates is the same: this is what we refer to as **domain repair** - often **domain abstraction** or **domain refinement**;

(ii) The number of arguments is different: this is **propositional abstraction** or **propositional refinement**;

(b) The names do not match:

(i) There is a type relation between the name of the predicate in the surprising question and the name of one of the preconditions: this is what we refer to as **predicate repair** - often **predicate abstraction** or **predicate refinement**;

(ii) There is no relation: this is **precondition refinement**.

In the examples given for the following cases, we are considering the action discussed in Example 1 and detailed in Equation 11. Unsurprising questions, in this context, would therefore be anything that matches the two preconditions of the rule: $(Submitted\text{-}Paper\ ?Paper\ ?Author\ ?Conference)$ and $(Format\ ?Paper\ ?Pdf)$ without any signature mismatches. When questions are put by the , as mentioned before, they may have uninstantiated variables, indicated by being preceded by a question mark as above, in which case not so much information can be inferred about the definition

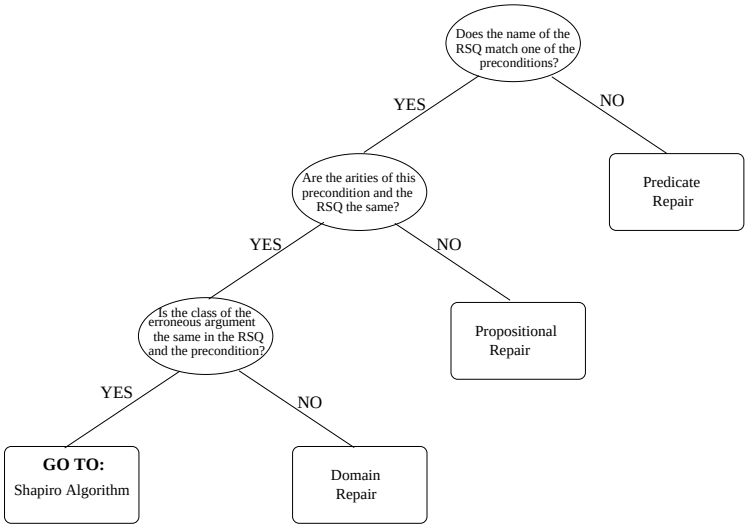


Figure 4. Diagnosis with Surprising Questions

of the predicate in the signature, or they may have some or all of the arguments instantiated, in which case type information about the arguments can be inferred.

Case a)i):

- Most of these situations conform to the following [3]:

$(p\ \vec{x}\ y)$ matches $(p\ \vec{x}\ z)$, $y \neq z$,
where p indicates a predicate, $\vec{x}$ indicates one or more arguments, and y and z are arguments.
There are four possibilities:

- $type(y) = type(z)$

In this case, the two predicates match with respect to the representational language, but conflict with respect to their specific instantiation. The source of this incorrect fact must be tracked down by the **Shapiro algorithm**.

- $type(y) \subset type(z)$ - domain refinement.
- $type(y) \supset type(z)$ - domain abstraction.
- $type(y) \nsim type(z)$.

$type(z)$ must already be in the ontology of , since z is a constant that is using in its planning. $type(y)$ may already be known by ; if not, the is queried to ascertain this value. Since the is using y in its communication, it must know what its type is.

Example 6 *The conversation proceeds as follows:*
: (Register Researcher Ai-Conf Registration-Cost)
: (Format MyPaper.pdf Doc)
: no.
The expectation of the was that the second argument would be Pdf but it is instantiated by the as Doc. The is fourth case: although there is a relation between the two types, it is not a simple sub- or super-relationship.

- Sometimes, a mismatch of this type may conform to the following:

$(p\ \vec{x}\ y\ z)$ matches $(p\ \vec{x}\ z\ y)$,
or, more generally,
$(p\ \vec{x}\ y\ z)$ matches $(p\ \vec{x}\ a\ b)$,
$type(y) = type(b)$ and $type(z) = type(a)$.
That is, the arguments have been transposed, either with the same instantiation or not.

Example 7 *The conversation proceeds as follows:*
: (Register Researcher Ai-Conf Registration-Cost)
: (Submitted-Paper ?Paper Ai-Conf Researcher)
: no.
Note that the first argument of Submit-Paper are uninstantiated as the does not yet know how it should be fulfilled. The expects the second argument to be of type Author and the third of type Conference and its facts are instantiated according to that; however, this order is reversed in the question put by the . It therefore cannot fulfil the expected question and responds negatively.

Case a)ii):

- $(p\ \vec{x}\ y)$ matches $(p\ \vec{x})$ - propositional refinement.

This signature repair is easy to diagnose and refine; all that is required is that $type(y)$ is ascertained, either through examination of the 's ontology or through questioning the .

- $(p\ \vec{x})$ matches $(p\ \vec{x}\ y)$ - propositional abstraction.

This is easy to diagnose and refine, both for the initial signature repair, and for the theory repairs that entails. It may be necessary to know what the type of the additional argument is, so that it is possible to tell which argument should be removed; this can be discovered either from the ontology or by asking the .

Example 8 *The conversation proceeds as follows:*
: (Register Researcher Ai-Conf Registration-Cost)
: (Submitted-Paper ?Paper ?Author)
: no.
The expects the predicate Submitted-Paper to have three

[3] Notation:
i)In all cases, the statement "*A* matches *B*" indicates that *A* is the surprising question put by the , and *B* is the question the expected, which corresponds with the 's ontology. Thus diagnosis and repair is aiming to change *B* so that it matches *A*;
ii) $\subset$ indicates subtype relation;
iii) $(p\vec{x}\ y)$ does not imply anything about the ordering of the arguments: it does not imply that y must be the last of the arguments.

arguments and so has no correct instantiation for the above query.

Case b)i):

- $(p\ \vec{x})$ matches $(g\ \vec{x})$

There are three possibilities:

- $type(p) \subset type(g)$ - predicate refinement.
- $type(p) \supset type(g)$ - predicate abstraction.
- $type(p) \nsim type(g)$.

This can be difficult to diagnose, because there may be insufficient information to determine that $(p\ \vec{x})$ should match precondition $(g\ \vec{x})$. If this information cannot be found, situations of this type are usually incorrectly classified as occurrences of case b)ii): a missing precondition. We cannot connect this surprising question to any of the preconditions, and so we assume that it is an additional precondition.

Example 9 *The conversation proceeds as follows:*
: (Register Researcher Ai-Conf Registration-Cost)
: (Submitted-Item ?Paper ?Author)
: no.
Submitted-Item could be considered to be a super-predicate of Submitted-Paper because Paper is a kind of Item.

Case b)ii):

- $(p\ \vec{x})$ fails to match any of the preconditions.

In this situation, we diagnose a missing precondition - precondition refinement - although, as discussed in case b)i), this will sometimes be incorrect. If it is incorrect, the effect will be to over-constrain the rule. We consider this to be an acceptable approach, because if this does occur, we are still left with a rule that will not be used other than in situations where it is correct to use it. The disadvantage is that there may be some situations in which it is correct to use it in which it appears to be unusable. However, since we consider that random name changing will not occur especially often, these situations will be rare, and a diagnosis of precondition refinement is usually correct.

Note that in , precondition abstraction can never be diagnosed: having unnecessary preconditions to satisfy can lead to unnecessarily complicated plans but does not lead directly to plan execution failure.

Example 10 *The conversation proceeds as follows:*
: (Register Researcher Ai-Conf Registration-Cost)
: (Money ?Author ?Amount)
: no.
There is no semantic link between Money and either of the two expected preconditions (Submitted-Paper and Format).

3.Failure after some questioning, with no surprising questions.

If failure occurs in this situation, we can conclude:

- Since questioning has begun, the only possible cause of failure is the 's response to these questions.
- The was expecting to be asked this question, and therefore ensured that it was correctly answered, as far as it understood this was to be done.

There are two ways in which this might happen:

(a) The last stage of the questioning immediately before failure proceeds as follows:
: : $(p\ \vec{x}\ ?Q)$
: : $(p\ \vec{x}\ A)$
That is, the is asking the to instantiate a variable $?Q$, and the instantiates this variable as A.

If such a response by the causes failure, then clearly the value A returned by the was unacceptable to the . This means that the 's expectations of what response would be appropriate are incorrect, and the 's preconditions need to be altered or added to so that they force A to be instantiated in the correct way. It is not difficult to diagnose this problem but it is usually difficult or impossible to repair the ontology correctly, since it is not know what values of A would have been acceptable to the . The best repair that can be done in the general case is to add an extra precondition stating explicitly that this variable $?Q$ should not be instantiated as A. However, this fails to extract any rules that might apply to the instantiation of $?Q$ and may well result in a rule that still instantiates this variable incorrectly. The other option is to mark the particular rule as unusable. This results in a more correct but less complete ontology and would be the preferable option if other rules were known that had the same or similar effects.

(b) The last stage of the questioning immediately before failure proceeds as follows:
:$(p\ \vec{x})$?
: *yes* (*or no*)
Here, the question from the is fully instantiated and the must respond as to whether it believes this statement is true or not. Diagnosis is therefore straightforward, as is repair, which is effected by negating the expectations of the truth value of this statement.

Example 11 *The conversation proceeds as follows:*
: (Register Researcher Ai-Conf Registration-Cost)
: (Format MyPaper.pdf Pdf)
: yes.
This is an expected question and, during planning, the has ensured that it is correct before the action takes place. However, if failure occurs immediately after this question, we can assume that the answer to this question was inappropriate. The only other possible answer would have been no. We therefore deduce that the 's expectations were incorrect and that it must ensure that (Format MyPaper.pdf Pdf) is not true (thought we don't, in this situation, have any information about what the format should be).

Dealing with Incorrect Facts: The Shapiro Algorithm and the Plan Deconstructor. In many cases, as has been discussed above, finding a problem fact will enable us to immediately diagnose what the problem is. For example, if we encounter a fact that contains an extra argument than we expect, it is clear that the problem is connected to this argument mismatch. However, in some cases, a fact is believed that is correct from a signature point of view but simply wrong, *i.e.*, incorrect from a theory point of view, such as occurs in Ex-

ample 4. This problem is thus not amenable to signature repair. Instead, we must discover how it was that this incorrect fact came to be believed.

As has been described above, there are many cases in which linking plan execution failure to specific mismatches in the ontology is straightforward because communication from agents reveal signature clashes. However, in cases where incorrect facts are believed, further analysis of the plan formation is necessary: we need to know how this fact came to be believed. The propositional and black-box nature of most modern planners, including the one used by (these issues are discussed further later in the paper) mean that it is not possible to extract directly from the planner any explanation of how the plan was built from the ontology. We have therefore developed a *plan deconstructor* which meta-interprets the plan with reference to the ontology to attach to each action the action rule that was used to justify it, the preconditions of that rule and the facts that have been made true or false or altered by its execution. The plan deconstructor acts in a similar manner to a first-order planner, but the massive search problems associated with such planners is removed as this has already been dealt with by an efficient propositional planner. For more details of these issues, see (McNeill, Bundy, Walton, & Schorlemmer, 2003). The plan deconstructor produces a justification for the plan, which is then used by the Shapiro algorithm.

The Shapiro algorithm traces back through the justification to determine where the value of the problem precondition was last changed. If this was a fact in the original ontology, it removes this fact. If this fact was an effect of a previous action, it checks with the agent that performed that action as to what it believes the value of the fact should be.

Summary of Diagnosis. Figures 2 - 4 (together with the algorithm for dealing with failure after unsurprising questioning, which is not represented graphically here) provide a full method for analysing the failure of an action during planning. In any circumstance, the algorithm is able to pin this failure down to a specific mismatch described in the space of possible mismatches, though in some cases it must be a guess and in some cases is not precise. It is not always possible, due to incomplete information, to use this diagnosis to implement a full repair to the ontology, though this is possible in many situations. However, these diagnostic algorithms provide a method for dealing with any possible cause of plan execution failure within the context of the problem as defined in this paper. Whether the context as defined here is broad enough to be useful remains to be seen: this issue is addressed in the evaluation section later in the paper.

Implementing repairs. Sometimes signature repair will be a matter of refining a single signature object. In such cases, implementing the diagnosis simply involves altering the ontology in the appropriate manner. In most cases, signature repair entails theory repair, because all theory objects written according to the original signature definition need to be altered so that they are now written according to the new definition. For example, if the definition of a predicate is changed so that it has one fewer argument, all instances of that predicate will also need to have the relevant argument removed so that the ontology remains consistent. In some cases, this theory repair will also be simple: in the above example of removing an argument, it can easily be determined, for every instance, which argument should be removed (for example, the second argument), and this can be implemented. In some cases, however, there is not enough information to repair theory objects fully. For example, if the signature repair was to add an argument to a predicate definition, then every instance of this predicate must also have an extra argument. It is possible to determine where in the instantiated predicate this argument should be added, and what type this argument should be, but it is not usually easy to determine what the specific instantiation of this argument should be. In such cases, our approach is to introduce *meta-variables* to such theory objects to act as place holders for these unknown arguments. It is then not possible to use these theory objects in planning.

A complete, automated solution to this problem that is guaranteed to be correct is impossible, since this missing information is not explicit in the ontology and cannot be automatically retrieved. The approach that takes to this problem is to infer information, where possible, from communication with other agents. For instance, in Example 1, an argument is added to the predicate *Money* of type *Payment_method*. The diagnostic algorithm can deduce, from the agent communication, that *Credit_card* is the required instantiation and so instantiates this unknown argument in this particular theory object accordingly. All other instantiations of the signature objects have meta-variables assigned to them, as their correct instantiations cannot be deduced. Circumstantial evidence can be useful here for deducing default instantiations. However, for these instantiations to be guaranteed correct, a human user of the ontology would need to view these meta-variables and decide how best to instantiate them.

Overview of Architecture

consists of various different sub-systems: the *translation system*, the *planning system*, the *agent communication system*, the *diagnostic system* and the *repair system*. The exists as part of the agent communication system and is able to call the other parts of the system as necessary. Figure 5 shows the interaction between the subsystems.

Flow of System

The system is controlled by the . The flow of control is illustrated in Figure 6. When a goal is passed to the system, the system reacts to this by translating the ontology into both (Fox & Long, 2003) and Prolog [4]. is a standard representation for planning and is used by many modern planners. The planner that uses is Metric-FF [5], which has won many awards and is widely accepted as a leading planner. is less expressive than : although it

[4] http://www.sics.se/sicstus/docs/latest/html/sicstus/

[5] http://www.mpi-sb.mpg.de/ hoffmann/ff.html

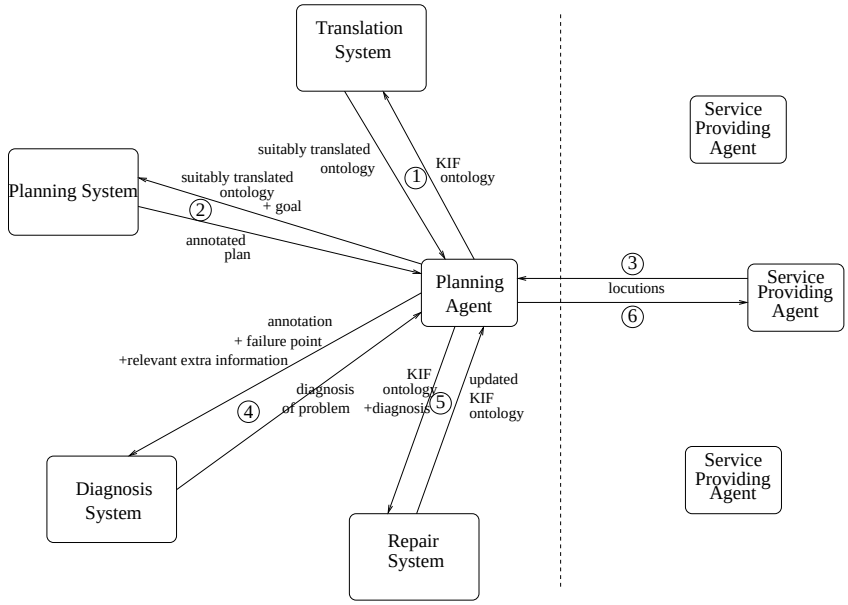


Figure 5. Architecture and Interaction of the

The planning agent ( ) calls upon modules for planning, diagnosis, repair and translation. The planner forms plans for specified goals, which is annotated with a justification of that plan with relation to the ontology. The then attempts to execute the plan by passing requests to the external agents. If this fails, the diagnostic algorithm is used to try to figure out what went wrong. Diagnosis may require further communication with the external agents. If a fault is detected then the repair module is asked to repair the ontology. A variety of representational formalisms are used by the different modules, so the translation module converts one formalism into another.

appears to be a first-order representation, it is in fact a first-order window onto a propositional domain. This difference in expressive power between the two representations may be considered as indicating that a planner such as Metric-FF is not appropriate for , but in fact this is a problem that is inherent and increasingly recognised in planning using real ontologies. Efficient, tractable planning cannot be done with a representation as rich as first-order logic, and yet standard planning representations are not expressive enough for representing ontological knowledge. Our approach to this problem is to use a rich representation for retaining knowledge and to translate this process to when planning is necessary. This ontology is then discarded when the plan has been formed and the translation is only one way. The differences in expressiveness are dealt with, for example, through the introduction of *pseudo-variables* to deal with the fact that cannot handle variables: the planner views these as constants and plans accordingly but the interprets the plan knowing that they represent variables. For further details of this translation process, see (McNeill, Bundy, & Walton, 2004). The agent platform is based on the Sicstus Prolog version of Linda (Patterson, Turner, & Hyatt, 1993), which is a set of language extensions based on a tuple-space. A server acts as a blackboard where agents, which are Linda clients, can write to, read from and delete messages. These agents therefore need a Prolog ontology, which is used during plan execution and can then also be discarded. The system also provides a translation process for this but this is simpler than the translation to because Prolog is also a first-order representation, so we do not discuss any of the details of this. If repairs are necessary, these are performed directly on the ontology. The and Prolog ontologies are not altered, since these have been discarded, and if a further goal is received, or the original goal is not yet reached, these will be regenerated from the new ontology.

Once the translation process has created the and the Prolog versions of the ontology, the then calls the planning system. This consists of the planner and the plan deconstructor, discussed earlier in the paper. The planner first forms a plan from the translated ontology and this plan is then deconstructed by the plan deconstructor, with reference to the ontology, and the plan, annotated with the justification, is returned to the .If the planner fails to return a plan, this is reported to the , and the process fails. This will occur if it is not possible to reach the goal from the initial state described in the ontology, using the actions described in the ontology. This may happen the first time that the agent attempts to form a plan, or it may be that it was initially possible to form a plan, but this plan failed to be successfully executed, and repairs made to the ontology, as a result of that failure, resulted in a situation where it was no longer possible to reach the goal.

The then attempts to execute the plan step by step. At each step, it locates the appropriate to perform the task,

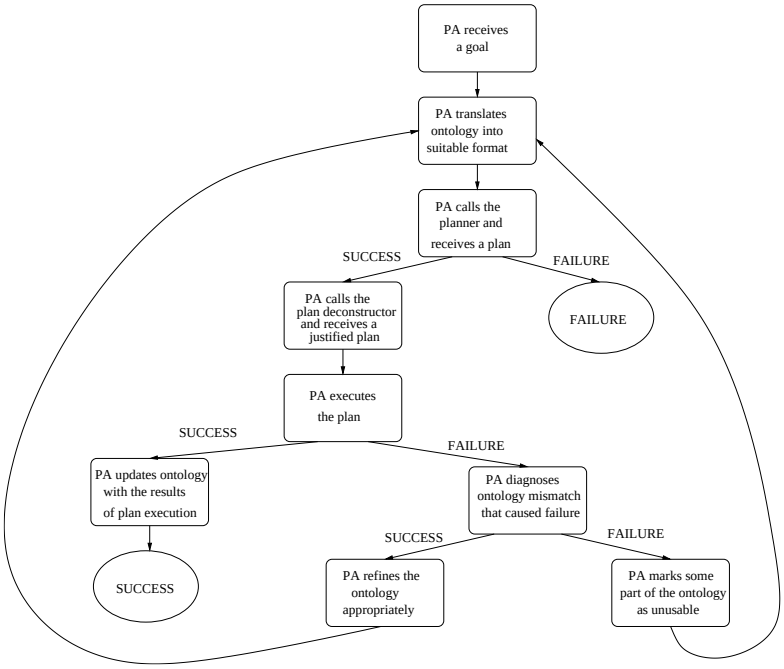


Figure 6. Flow of Control of the

The translates the ontology, sends it to the planner and receives the plan. If this fails, the process terminates. If it succeeds, the attempts to execute the plan. If this ends in success, the process terminates. If any plan step fails to be executed, the attempts to diagnose the mismatch. If this is performed successfully, the ontology is altered accordingly and retranslation and replanning is recommenced. If it fails, the offending part of the ontology is marked as unusable and translation and planning begins again without it.

sends a request to that agent to perform the task and waits for one of the following responses:

1. Information about the final outcome:

• An indication that the action has been successfully performed;

• An indication that the action has failed to be performed;

2. A request for further information:

• A fully instantiated predicate, for which the must give the truth value according to its ontology; an agent will either reply that this is true if it has information to indicate this, or that it cannot confirm the truth if it does not have such information;

• A fully or partially uninstantiated predicate, which the must instantiate according to its ontology. If there is no fact in its ontology that corresponds with this, it replies that this is false.

If the response received from the is of type 2, the , after responding appropriately to it, waits again for one of the above responses. This loop continues until a response of type 1 is received. If the action succeeds, the then attempts the next step of the plan, until the goal is reached. At this point, the returns output to indicate that the plan has succeeded, and terminates. If all the actions succeed, the system is just performing planning within a multi-agent system. It is when failure occurs that the core abilities of are called into use: diagnosis and repair. If any action fails, the calls the diagnostic system to receive a diagnosis of the problem. A diagnosis is always returned: either a precise diagnosis, if sufficient information is available to discover it, or else a diagnosis that one or more ontological objects should not be used in planning, as they are faulty in some unknown way. The repair system is then called to implement the diagnosis. The process is then repeated. The updated ontology is retranslated, and a new plan is formed, if possible: the refining of the ontology could mean that it is impossible to form a plan to achieve the goal. If a plan is produced, this is then executed step by step in the manner described above. The process is repeated until the goal is achieved, or until diagnosis, repair or replanning proves impossible.

Complexity of System

The application of the diagnostic algorithm consists of a single traversing of the fault tree. It is never necessary to retrace steps in the fault tree as we can determine at each step which the best route to take is and even if we end in an uncertain diagnosis, we can be sure that this is the best diagnosis available with the given information and that no retraversing of the fault tree will be efficacious. Therefore, if we assume that the decision point at each node is in constant time, diagnosis by fault tree is in constant time.

The assumption that the decision at each node takes con-

stant time is certainly not always valid; such decisions usually involve interaction with other agents and thus the amount of time they take depends both on the communication infrastructure and the amount of time another agent takes to process its response. However, such operations are outside our control and we can therefore only give complexity estimates for our algorithm dependent on such unknown constraints.

The exceptions to this are the cases in which the diagnostic algorithm leads to the calling of the Shapiro algorithm. The application of this consists of tracing back through the plan formation one step at a time and, at each step, checking the effects of that step to determine whether the relevant fact was altered by it. It is therefore linear in the length of the plan.

Other aspects of the system have much larger time complexity than this: for example, that of the planner is exponential. We therefore only discuss the complexity of the central diagnostic aspects of the system.

Evaluation

The nature of our system as illustrated in Definition 2 means that standard methods of evaluating ontology matching systems, such as giving them as input two mismatched ontologies and determining how many of these mismatches they discover and how long it takes them to do this, are irrelevant. is not designed to deal with ontology mismatches in the abstract – to be able to discover any mismatch between two given ontologies – but to track down specific mismatches that are causing problems in an interaction. In order to evaluate it, therefore, we must develop these interaction situations and then establish that the system can, in fact, track down any mismatches that cause problems. Additionally, the minimal approach of , where mismatches are ignored except when they are discovered to be causing problems, means that we cannot simply input two ontologies to and see how many of the mismatches between them can discover. is intended to ignore mismatches that do not lead to communication problems, and we therefore need to set up this communication environment where mismatches between agents actually lead to plan execution failure to establish how well performs. This means that it is only possible to evaluate it in a situation in which we have disparate agents interacting in a multi-agent system, with one agent acting as and one or more acting as . In order to determine how performs, we must have agents with ontologies from which they can form plans to achieve a given goal and, in the execution of these plans, interact with agents who have ontologies which are not only slightly mismatched to theirs but with mismatches that occur in relevant places, so that these particular actions bring them to light. Different versions of large ontologies tend to have a large number of mismatches between them; however, it is difficult to test against all of them, as this means we need to develop an interaction situation in which each mismatch leads directly to a problem. Two mismatched ontologies are not sufficient input to evaluate ; we must also allow to operate in an interaction situation in which mismatches may come to light.

We therefore divided the evaluation of into two parts. The first part of the evaluation was to determine whether the algorithm was able to actually diagnose and repair the mismatches that it is intended that it should: that is, does the algorithm work correctly? This was performed with ontologies created from different versions of real-world ontologies as well as other ontologies we developed ourselves, so that every kind of possible mismatch was tested. The purpose of this part of the evaluation was to determine that every kind of mismatch that we claim can diagnose and repair did prove, in practice, to be diagnosable and repairable by . This element of the evaluation therefore provided an environment in which we could test whether our algorithm succeeded or failed at diagnosing and repairing mismatches that it is intended to be able to perform, and through this evaluation we can determine whether the algorithm is successful. However, in order to determine the real value of the algorithm, it is also necessary to determine whether the diagnostic techniques that are embodied in the algorithm are really the ones which are encountered in real life. We already know areas in which the algorithm will fail: adding ontological objects is sometimes impossible because there is insufficient information; removing ontological objects is usually impossible because their presence does not lead to plan execution failure and thus cannot be discovered by the algorithm; certainly any ontological object that is not part of an ontology built according to Definition 4 cannot be correctly diagnosed by . The degree of failure that these imply differs according to the type of failure: for example, a failure due to the ontology not conforming to our definition of an ontology is not really a failure of the algorithm itself but rather a failure of scope of the context of the problem. However, if a large proportion of encountered mismatches are not those that are diagnosable by the algorithm, then its value must be questioned. The second part of the evaluation was therefore to determine, for large real world first-order ontologies available in different versions, how many of the mismatches between them would be diagnosable by . We therefore divided them into categories determining which, if they were the cause of plan execution failure, would be diagnosable and repairable by and which would not (and, for such mismatches, why not). The first part of the evaluation has already verified that, for all kinds of mismatches that are classified as being one that can diagnose and repair, can actually do this. We present below the statistics derived from these results and discuss what they mean: which failures point to flaws in the theory, which to limitation of scope and which can be considered irrelevant.

In undertaking the first stage of the evaluation, we first tested the performance of the system on seven different ontologies, where the had a slightly different version to the other agents with which it was interacting. Three of these were based on off-the-shelf ontologies: , and . These ontologies were altered so that their format matched the expected input format for the system and so that there was some kind of planning format overlaid on them: all of these ontologies (like most publicly available ontologies) are static and do not include action rules or many individuals and

thus cannot immediately be used for planning. However, we believe that this imposition of a planning context does not lessen the value of these results because this does not affect the ontological mismatches that is diagnosing and repairing; it is simply adding extra ontological objects, such as individuals and action rules, so that these mismatches can form part of the communication between the agents. These three ontologies are all first-order and written in or a similar representation. However, none exactly matched the restricted used by and they therefore had to be altered slightly to be used as input to the system. This merely involved removing the detail from complex class definitions and ignoring ontological objects that contained quantification. It did not involve changing the mismatches we were testing. The other four were based on planning scenarios taken from the (European Network of Excellence) repository [6] but we did not have different versions of these ontologies available and had to create the mismatches between these ontologies. We were able to demonstrate that could successfully perform all the mismatches described earlier in this paper: see the project webpage [7] for details.

When considering the second part of the evaluation, we define four different categories into which mismatches may fall, one of which is further subdivided:

1. could refine the mismatch because the mismatch fell into one of the categories we have demonstrated can deal with: *e.g., changing the arity of a predicate was appropriate*;
2. could not currently refine the mismatch, but minor changes to the system would allow to refine it: *e.g., changing the class of an instance*;
3. could not refine the mismatch. This is because:

(a) did not have sufficient functionality: *e.g., double implication was altered to single implication*;

(b) This particular mismatch is outside the scope of the project: *e.g., a compound mismatch*;

(c) This mismatch is irrelevant to an automated system: *e.g., a change to commenting or formatting*;

(d) This mismatch could not occur in the restricted that is designed to use: *e.g., a change to a complex class definition*;

(e) This mismatch could not be highlighted in a planning context: *e.g., an instance is removed from the ontology*.

4.. The information we had about the mismatch was insufficient to diagnose which of the above categories it would fall into.

From the above categories, it is clearly desirable that as many as possible fall into category 1; such mismatches indicate a successful outcome for . However, given the assumptions we have had to make in and the unsuitability of the ontologies against which it is evaluated, it is to be expected that many will fall into categories 3b- 3e. A poor outcome for would be represented by many mismatches falling into category 3a. It may seem fairly arbitrary whether mismatches are assigned to category 3a or category 3b: is a mismatch unrefinable because it is outside the scope of the project, or because does not have sufficient functionality? We assign mismatches to 3b if they belong to categories that we have explicitly ignored: compound mismatches and random changes; otherwise we assign them to 3a. Under such circumstances, mismatches being assigned to 3a is the worst outcome; nevertheless, a large number of mismatches being assigned to 3b would indicate that is not especially well adapted to the task it is attempting to perform. A large number of mismatches in 3d does not directly provide a poor outcome for , since we have made and justified a decision to limit the ontologies with which can operate.

We summarise our results in Figure 7, where the proportions of mismatches that fall into each category are illustrated. The complete ontologies from which these results came, and the complete set of their mismatches, can be found through links from the project website. The mismatches on the website are highlighted to illustrate which category each mismatch falls into.

Analysis of Results

The results illustrated in the pie-chart indicate a reasonably good performance for : it can perform 38.8% of all mismatches. If we consider only *significant* mismatches, that is, ignoring categories 3c (alterations to commenting and other changes that are not directly to the ontology) and category 4 (which are impossible to evaluate), we find that can perform 45.0% of the mismatches. If we consider only *relevant* mismatches, that is, ignoring categories 3d and 3e, which are specifically outside the scope of the system, we find that can perform 70.8% of the mismatches. Here, we discuss the kinds of repairs that fall into each of the categories.

- **Category 1** *(could refine the mismatch)*

The evaluation illustrates that a wide range of the potential mismatches we identified can actually be found in these ontologies. Additionally, we see that a fairly high proportion of mismatches fall into this category, despite the fact that these ontologies are written in a different representation to the one required for , are not designed for planning and have been altered on the assumption that these changes will be interpreted by human users.

- **Category 2** *(could refine the mismatch after small changes)*

Only 6.5% of mismatches fell into this category. All of these fell into three different groups:

- Changing the super-type of a type;
- Changing the type of an instance;
- Adding relations to the ontology:

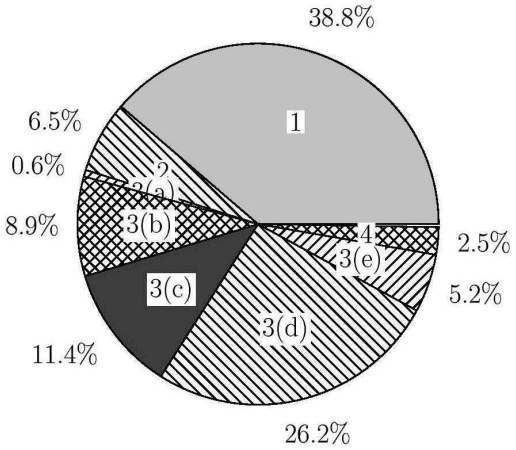
- **Category 3a** *(did not have sufficient functionality)*

The results in this category are very low, which is a good result for . In fact, only two of all the mismatches fell into this category.

- One was concerned with a double implication in a rule being altered to one-way implication: does not deal with double implication because this is not standard in action rules.

[6] http://scom.hud.ac.uk/planet/repository/

[7] http://dream.inf.ed.ac.uk/projects/dor/



Each segment of the piechart represents and is labelled by one of the mismatch categories defined in the above classification. Its size represents the proportion of examples of that kind of mismatch found in the ontologies we analysed.

Figure 7. Results of the Evaluation of

• An individual was altered to be a type. does not consider mismatches between different types of ontological objects.

• **Category 3b** *(the mismatch is outside the scope of the project)*

The percentage of mismatches that fall into this category is far larger than the percentage of those that fall into the previous category: nearly 9% of all mismatches, fell into this somewhat undesirable category. We consider these mismatches to be outside the scope of the project because they fall into one of two categories:

• Multiple interacting mismatches;

• Arbitrary changes to the ontology.

• **Category 3c** *(the mismatch is irrelevant to an automated system)*

Mismatches that fall into this category are not really interesting to the evaluation. However, the fact that they occur reasonably frequently (11.4% of all mismatches) indicates that there is much information in these ontologies which is not readable automatically by agents that cannot interpret natural language, and that some effort is put into updating this information. This is a reflection of the assumption that this information is relevant only to human users. If automated systems perform this task more commonly, effort will be put into explaining these changes using ontological objects where possible, rather than through commenting.

• **Category 3d** *(the mismatch could not occur in restricted)*

This category is relatively common, largely due to the fact that the complex class definitions that we exclude from our representation occur fairly frequently.

• **Category 3e** *(the mismatch could not be highlighted in a planning context)*

These mismatches are exclusively concerned with objects being removed from the ontology.

• **Category 4** *(insufficient evidence to properly classify mismatch)*

Related Work

It can be seen from Definitions 1 and 2 that our work is fundamentally different to that of traditional ontology matching, and the comparison between the two approaches should be clear from the introduction to this paper. We therefore do not repeat that discussion in this section; see (Campbell & Shapiro, 1998; Wiesman, Roos, & Vogt, 2002; Doan, Madhavan, Dhamankar, Domingos, & Halevy, 2003; Giunchiglia & Shvaiko, 2003; Kalfoglou & Schorlemmer, 2003; Giunchiglia, Shvaiko, & Yatskevich, 2005) for further details about this approach.

We believe that this pragmatic, piece-meal, agent-centric approach to ontology matching is novel; however, there has been work on structural mismatch in other contexts. (Visser, Jones, Bench-Capon, & Shave, 1998) presents a method of assessing heterogeneity between ontologies and is concerned not just with semantic but also with structural mismatches. However, this work, like most ontology matching, looks at the problem of comparing two complete ontologies before interaction commences and is interesting only in assessment of heterogeneity rather than in matching and repairing ontologies. (Klein, 2001) provides an analysis of problems and solutions in the field of combining and relating ontologies, some of which overlaps with our interests. However, the approaches mentioned are again interesting in aligning entire ontologies and are not fully automated. The scope of the approaches is much broader than our scope: for example, mismatches between different languages are examined, and this is possible because a much higher level of human involvement is assumed.

In the field of database integrity, the problem of incomplete information and inconsistency can lead to difficulties in the task of query processing, and much work has been done in this area (Cali, Calvanese, Giacomo, & Lenzerini, 2002). Work on maintaining the consistency of databases after data has been integrated from multiple sources usually focuses on tuple insertion and deletion (Greco, Greco, & Zumpano, 2003). However, some work has been done on *value modification*, where tuples referring to similar things but structured and labelled differently are merged (Bohannon, Fan, Flaster, & Rastogi, 2005). This is in some ways similar to our work on ontology alignment; however, this work depends on pre-defined *inclusion dependencies*, which detail how a schema from one database matches a schema from another database, as well as *functional dependencies*, which detail dependencies within a schema, and these are key in determining matches and repair: this has no parallel in ontology alignment. Additionally, there is full access to both (all) databases and (manually derived) weights giving information as to how accurate each datum is believed to be. (Wijsen, 2003) also deals with the problem of inconsistent tuples in databases, focusing on repairing them so as to retain as much pertinent information as possible rather than deleting them from the database to maintain consistency. A difference from our work is that the target tuple is not clear: there may be many ways of altering a tuple so that it becomes consistent. In our case, however, there is a clear target – the representation of the other agent – which we are aiming towards. This work is also concerned with inappropriate instantiations of tuples that violate integrity constraints (for example, one date argument ought to represent a later time than a second date argument and does not) and does not address, as we do, mismatches in the underlying signature (for example, the argument ought not to be a date at all).

The field of constraint mapping and dynamic query translation focuses on an area that is to some extent similar to our work. However, constraint mapping, like much ontology matching, tends to focuses on concept mapping, and in addition makes use of specific user-provided mapping rules (Chang & García-Molina, 1999), and thus cannot be considered to be a general, fully-automated approach as is. Although in general this work does not deal with predicate mapping, (Z.Zhang, B.He, & K.C.-C.Chang, 2005) introduces an attempt to provide this functionality. However, this work is domain specific rather than general, and the predicates involved are rather different to our notion of predicates: they are really query templates. Additionally, there is also a clear source and target predicate, so that it is not necessary, as in to search for an appropriate predicate to match to.

Another interesting parallel is work on plan repair in multi-agent systems (Krogt & Weerdt, 2005a, 2005b). Like our work, this focuses on executing plans in a multi-agent scenario with limited access to information, and on dealing with plan execution failure when it inevitably occurs. Like , this work considers the need to alter preconditions and effects of action rules and the removing of propositions already in the agent's knowledge base (or ontology) in light of new information from the world. However, they do not consider the central focus of : representational change when it is discovered that the current representation is not appropriate to describe the world. We also do not tackle the problem of plan repair: instead, repairs the ontology and then plans from scratch using the new ontology. Incorporating some of this work on plan repair into may lead to a more efficient system.

Further Work

is intended to be a first approach to the problem defined in the paper. As such, there are many assumptions and simplifications made so that we can reduce this complex problem to something that is tractable. The further work we wish to do with involved removing these assumptions and simplifications, thereby making the system more applicable to an environment such as the Semantic Web.

This work would include:

- Investigating the use of full with , rather than the restricted we are currently using, and investigating its applications to other ontological representations such as . A difficulty here would be whether these other representations would be rich enough for the planning environment we are interested in.
- Investigating a more sophisticated approach to implementing the repairs on the ontology. Perhaps altering the ontology and disposing of the old one is not always the best approach: different ontologies may be appropriate in different situations and with different agents; older versions may prove to be more correct and could be reverted to, and so on.
- Ontological mismatches could be negotiated between agents, rather than the being always assumed to be in the wrong.
- Diagnostic restrictions could be relaxed, such as the fact that we do not deal with compound mismatches.
- Plan repair based in conjunction with ontological repair may prove more efficient than replanning from an improved ontology; this should be investigated.
- focusses almost exclusively on structural mismatches (where terms are composed differently) rather than semantic mismatches (where individual words are used differently). We made this decision because of the large body of work that has already been done in semantic matching. However, to make practically useful, we need to incorporate work already done on semantic matching so that can match terms that are both structurally and semantically different. We have begun some initial investigations into the idea of amalgamating the abstraction and refinement structural mismatches identified in with state-of-the-art semantic matching (Giunchiglia, Yatskevich, & McNeill, 2007); however, we have not yet extended this to be incorporated in itself.

Conclusion

In this paper, we presented our system, , the theory underpinning it and some experimental results. We believe that not only is our approach novel in the ontology matching field but that our interpretation of the what the problem is

also novel. is designed as a system to be used by an agent attempting to use its ontology to help it interact with the world. The expectation is that the ontology is an accurate and full representation of that world, but this expectation is almost never valid. In reality, interaction with the world will repeatedly highlight flaws in the ontology. When these flaws lead to failure to achieve goals, such agents can call on to use whatever information can be gleaned from the interaction to diagnose and repair the ontological fault that led to this interaction failure, and to then replan with this improved ontology and attempt once again to reach the goal. We have presented evidence of how performs against genuine mismatched ontologies and demonstrated that such ability is a major advantage in communication between disparate agents with mismatched ontologies. We view as a first step on the path and have discussed in this paper the ways in which this work could be extended to provide a more complete solution to the problem.

References

Bohannon, P., Fan, W., Flaster, M., & Rastogi, R. (2005). A cost-based model and effective heuristic for value-based constraint repair. In *Proceedings of the 2005 special interest group on management of data.* Baltimore, Maryland, USA.

Bundy, A., & McNeill, F. (2006). Representation as a fluent: An AI challenge for the next half century. *IEEE Intelligent Systems*.

Cali, A., Calvanese, D., Giacomo, G. D., & Lenzerini, M. (2002). Data integration under integrity constraints. In *Proceedings of the 14th conf. on advanced information systems engineering (caise 2002).*

Campbell, A. E., & Shapiro, S. C. (1998). Algorithms for ontological mediation. In *Proceedings of the COLING-ACL workshop* (pp. 102–107). Montreal, Canada.

Chang, C.-C. K., & García-Molina, H. (1999). Mind your vocabulary: query mapping across heterogeneous information sources. In *Proceedings of the 1999 special interest group on management of data* (pp. 335–346). Philadelphia, Pennsylvania, USA.

diSessa, A. (1983). Phenomenology and the evolution of intuition. In A. Stevens & D. Gentner (Eds.), *Mental models* (p. 15-33). Erlbaum.

Doan, A., Madhavan, J., Dhamankar, R., Domingos, P., & Halevy, A. (2003). Learning to match ontologies on the semantic web. *The VLDB Journal*, *12*(4), 303–319.

Fox, M., & Long, D. (2003). PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of AI Research*, *20*, 61-124.

Genesereth, M. R., & Fikes, R. E. (1992). *Knowledge Interchange Format, Version 3.0 Reference Manual* (Tech. Rep. No. Logic-92-1). CA, USA: Stanford.

Giunchiglia, F., & Shvaiko, P. (2003). Semantic matching. In F. Giunchiglia, A. Gomez-perez, A. Pease, H. Stuckenschmidt, Y. Sure, & S. Willmott (Eds.), *Workshop on ontologies and distributed systems (ods 2003).* Acapulco, Mexico.

Giunchiglia, F., Shvaiko, P., & Yatskevich, M. (2005). S-match: an algorithm and an implementation of semantic matching. In Y. Kalfoglou, M. Schorlemmer, A. Sheth, S. Staab, & M. Uschold (Eds.), *Semantic interoperability and integration.* Internationales Begegnungs- und Forschungszentrum (IBFI), Schloss Dagstuhl, Germany.

Giunchiglia, F., & Walsh, T. (1990). The use of abstraction in automatic inference. *Proceedings of UK Conference on Information Technology (IT-90)*.

Giunchiglia, F., Yatskevich, M., & McNeill, F. (2007). *Structure preserving semantic matching* (Tech. Rep. No. EDI-INF-RR-0955). University of Edinburgh: Informatics Research Report.

Greco, G., Greco, S., & Zumpano, E. (2003). A logical framework for querying and repairing inconsistent databases. *IEEE Transactions on Knowledge and Data Engineering*, *15*(6), 1389–1408.

Hayes-Roth, F. (1983). Using proofs and refutations to learn from experience. In *Machine learning* (pp. 221–240). Palo Alto, CA: Tioga Publishing.

Kalfoglou, Y., & Schorlemmer, M. (2003). If-map: an ontology mapping method based on information flow theory. *Journal on Data Semantics*, *1*(1), 98-127.

Klein, M. (2001). Combing and relating ontologies: an analysis of problems and solutions. In A. Gomez-Perez, M. Gruniger, H. Stuckenschmidt, & M. Uschold (Eds.), *Proceedings of ijcai workshop on ontologies and information sharing.* Seattle, USA.

Krogt, R. van der, & Weerdt, M. de. (2005a). Coordination through plan repair. In M. Gleizes, G. Kaminka, A. Nowe, S. Ossowski, K. Tuyls, & K. Verbeeck (Eds.), *Proceedings of the third european workshop on multiagent systems (eumas)* (pp. 509–510). Koninklijke Vlaamse Academie van Belgie voor de Wetenschappen en Kunsten.

Krogt, R. van der, & Weerdt, M. de. (2005b). Coordination through plan repair. In Gelbukh, de Albornoz, & Terashima-Marin (Eds.), *Micai 2005: Advances in artificial intelligence* (Vol. 3789, pp. 264–274). Springer.

McNeill, F., Bundy, A., & Walton, C. (2004). An automatic translator from KIF to PDDL. In *Proceedings of the 23rd workshop of the UK planning and scheduling special interest group (PlanSIG 2004).* Cork, Ireland.

McNeill, F., Bundy, A., Walton, C., & Schorlemmer, M. (2003). Plan execution failure analysis using plan deconstruction. In *Proceedings of the 22nd workshop of the UK planning and scheduling special interest group (PlanSIG 2003).*

Patterson, L. I., Turner, R. S., & Hyatt, R. M. (1993). Construction of a fault-tolerant distributed tuple-space. In *Proceedings of the 1993 ACM/SIGAPP symposium on appliced computing: states of the art and practice.* Indianapolis, United States.

Shapiro, E. Y. (1982). *Algorithmic program debugging*. The MIT Press.

Visser, P., Jones, D., Bench-Capon, T., & Shave, M. (1998). Assessing heterogeneity by classifying ontology mismatches. In *Proceedings of the 1st international conference on formal ontology in information systems (FOIS)* (pp. 148–162). Trento, Italy.

Wiesman, F., Roos, N., & Vogt, P. (2002). Automatic ontology mapping for agent communication. In *Aamas '02: Proceedings of the first international joint conference on autonomous agents and multiagent systems* (pp. 563–564). New York, NY, USA: ACM Press.

Wijsen, J. (2003). Condensed representation of database repairs for consistent query answering. In *Proceedings of the 9th international conference on database theory (ICDT 2003).* London, UK: Springer-Verlag.

Z.Zhang, B.He, & K.C.-C.Chang. (2005). Light-weight domain-based form assistant: Querying web databases on the fly. In *Proceedings of the 31st very large databases conference (vldb 2005)* (pp. 97–108). Trondheim, Norway.